Laurea Triennale in informatica - Università di Salerno
Corso di *Ingegneria del Software* - Prof. Carmine Gravino

ODD
Object Design Document
Sapori di Unisa

Riferimento	nc01_saporidiunisa-odd.docx
Versione	1.0
Data	01/02/2024
Destinatario	Prof. Carmine Gravino
Presentato da	Antonio Facchiano, Gianmarco Riviello, Salvatore Ruocco, Simone Vittoria
Approvato da	

Revision History

Data	Versione	Descrizione	Autori
20/12/2023	0.1	Introduzione	Salvatore Ruocco
22/12/2023	0.2	Packages	Salvatore Ruocco
03/01/2024	0.5	Class Interfaces	Gianmarco Riviello, Simone Vittoria
06/01/2024	0.6	Attributi	Antonio Facchiano
29/01/2024	0.7	Revisione Class Interfaces, Glossario	Gianmarco Riviello
29/01/2024	0.8	Design Patterns	Gianmarco Riviello Salvatore Ruocco
31/01/2024	0.9	Entity diagram	Simone Vittoria
01/02/2024	1.0	Revisione e delibera del documento	Facchiano Antonio; Riviello Gianmarco; Ruocco Salvatore; Vittoria Simone

Sommario

1 Introduzione

Sapori di Unisa si propone di semplificare e ottimizzare attività quotidiane di un supermercato e di conseguenza ridurre il più possibile gli errori umani.

Il documento di Object Design ha come obiettivo quello di produrre un modello che sia in grado di integrare in modo coerente e preciso tutte le funzionalità individuate nelle fasi precedenti. Inoltre, verranno definiti i packages, le interfacce delle classi e verrà ristrutturato il class diagram realizzato durante l'analisi dei requisiti.

1.1 Object Design Goals

Modularità
Il sistema deve basarsi su alta coesione e basso accoppiamento tra le sue componenti.

Robustezza
Il sistema dovrà garantire robustezza reagendo correttamente a situazioni impreviste usufruendo di opportuni controlli degli errori e gestione delle eccezioni.

Incapsulamento
Il sistema deve limitare l'accesso diretto alle sue componenti *(Information hiding*).

Analisi trade-off

Facilità d'uso vs Complessità
L'introduzione di funzionalità aggiuntive più specifiche rispetto a quelle già progettate, accrescerebbe la complessità del sistema rendendolo più difficile da utilizzare per gli utenti meno esperti. Si è scelto così di dare priorità alla facilità di utilizzo del sistema.

Gestione dei fallimenti vs Tempi di risposta
Avere procedure troppo complesse di gestione dei fallimenti potrebbe rallentare la risoluzione di problemi e influire sui tempi di risposta. Motivo per cui, si è deciso di effettuare un'accurata gestione dei fallimenti ma senza influire negativamente sulle performance del sistema.

Riservatezza vs Condivisione dei dati
Per garantire una giusta sicurezza ai diversi utenti che usufruiscono del sistema, si è deciso di effettuare una netta distinzione tra le operazioni concesse ad ognuno di essi. Tutto ciò a discapito della comunicazione e della condivisione dei dati tra gli utenti.

Componenti off-the-shelf

Per il progetto software che si vuole realizzare, facciamo uso di componenti off-the-shelf. Sono componenti software disponibili sul mercato che servono a facilitare il lavoro di molti developers. Una di queste componenti è **jQuery** (3.7.1); jQuery è una libreria JavaScript, molto utile per le applicazioni web perché permette di rendere il codice più sintetico; infatti, il suo motto è *«write less, do more.»*.

Altra componente che verrà utilizzata è **AJAX**, acronimo di Asynchronous JavaScript and XML. AJAX è un insieme di tecniche e metodologie di sviluppo software per la realizzazione di applicazioni web interattive. Si basa su uno scambio di dati in background fra web browser e server, consentendo così l'aggiornamento dinamico di una pagina web senza esplicito ricaricamento da parte dell'utente.

Come formato per lo scambio dati tra una servlet ed una pagina web in modo dinamico, si è deciso di utilizzare **JSON** (*JavaScript Object Notation*); esso è un formato di scambio dati leggero, facile da leggere e scrivere.

Per migliorare la leggibilità del nostro codice, abbiamo pensato di utilizzare la libreria **Lombok**, la quale permette di limitare la scrittura di codice ripetuto attraverso le annotazioni.

Per la creazione di grafici a torta ci siamo affidati alla libreria grafica JavaScript **Charts.js**.

1.2 Linee guida per la documentazione dell'interfaccia

È richiesto agli sviluppatori di seguire le seguenti linee guida al fine di essere consistenti nell'intero progetto e facilitare la comprensione delle funzionalità.

Di seguito una lista di link alle convenzioni usate per definire le linee guida:

- **W3Schools – HTML Style Guide**
- **Google Java Style**

Per adattare le linee guida selezionate nell'IDE di sviluppo il team prevede di utilizzare il plug-in **Checkstyle**. Come base verrà utilizzato l'insieme di regole fornite da Google all'interno della sua configurazione iniziale per il plug-in.

Per avere una documentazione corretta e completa adottiamo lo strumento **Javadoc**.
Per ogni classe devono essere definiti:

- Autore
- Descrizione

Per ogni metodo devono essere definiti:

- Descrizione
- Parametri in input
- Tipo di ritorno
- Eccezioni

1.3 Definizioni, acronimi e abbreviazioni

- **Package:** raggruppamento di classi, interfacce o file correlati.
- **Design pattern:** template di soluzioni applicati a problemi ricorrenti, utili per avere riutilizzo ed espandibilità del sistema.
- **Interfaccia:** insieme di prototipi di metodi offerti dalla classe.
- **Javadoc:** applicativo utilizzato per generare automaticamente documentazione al fine di renderla facilmente accessibile e leggibile.
- **Camel case**: in italiano “notazione a cammello”, consiste nello scrivere parole tralasciando gli spazi, ma con le iniziali maiuscole.

1.4 Riferimenti

- Libro di testo utilizzato durante il corso: **Object-Oriented Software Engineering Using UML, Patterns, and Java**
- **Statement Of Work**
- **Requirements Analysis Document**
- **System Design Diagram**

2 Packages

In questa sezione del documento mostriamo la suddivisione in package del sistema; tale suddivisione riprende l'organizzazione definita da Maven.

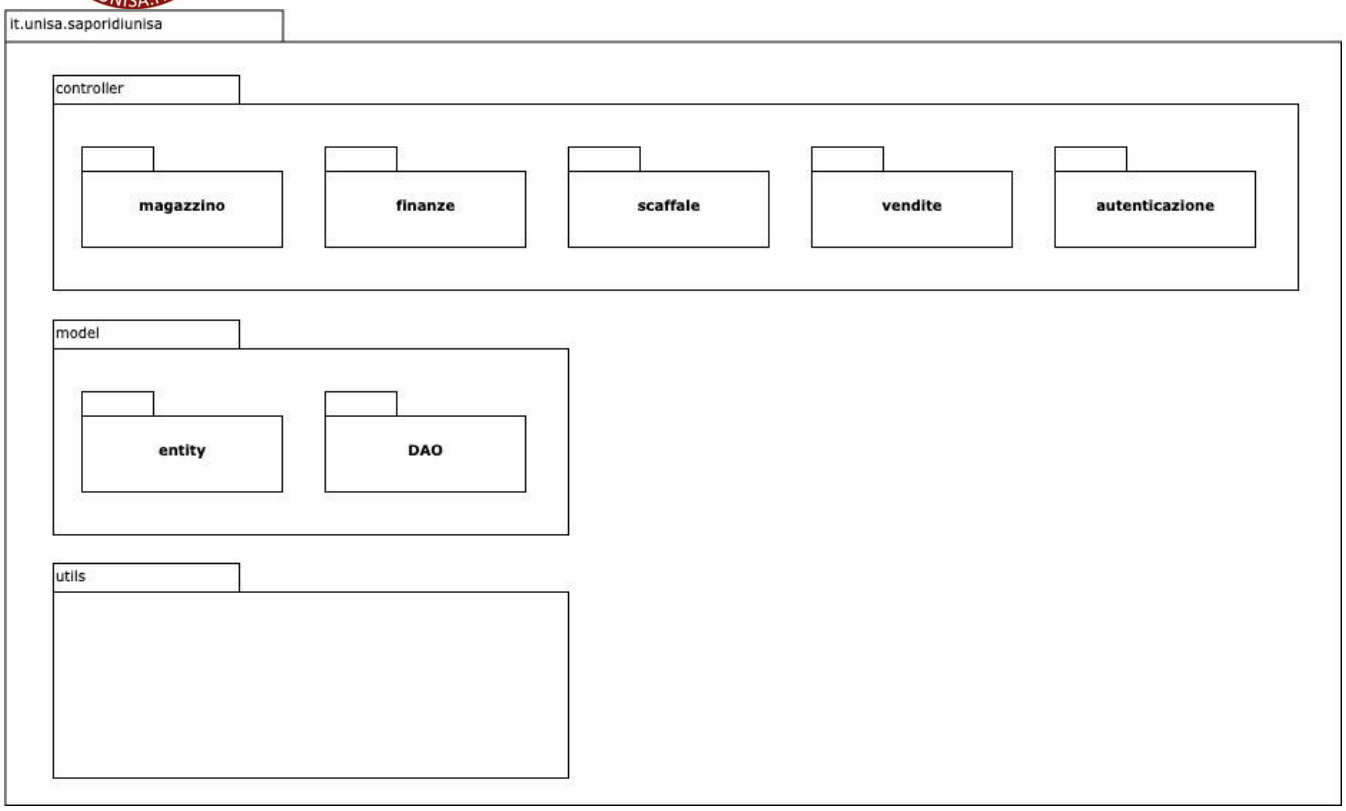
- **.idea**: configurazione di IntelliJ
- **.mvn**: configurazione di Maven
- **src**: codice sorgente
    - **main**
        - **java**
            - **it.unisa.saporidiunisa**
        - **resources**: contiene gli assets
        - **Webapp**
            - **img**
            - **style**
            - **script**
            - **view**
            - **WEB-INF:** file non accessibili dai client in modo diretto
    - **test**
        - **java:** classi java per il testing
    - **target:** file compilati

Package Sapori di Unisa

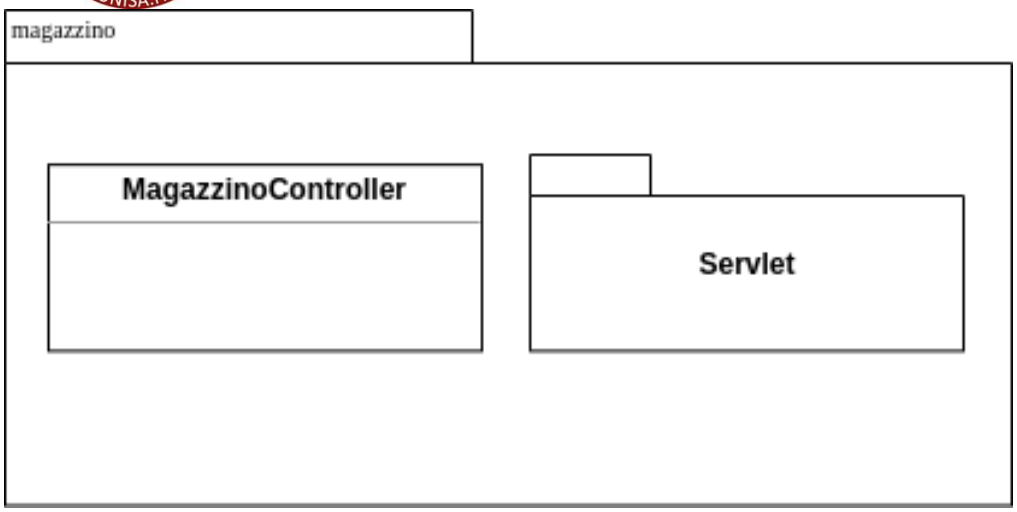
Il package principale di Sapori di Unisa **it.unisa.saporidiunisa** contiene:

- **controller**: contiene i package dei sottosistemi già individuati nel documento di System Design.
- **model:** contiene il package entity e il package DAO.
- **utils**: contiene classi utilizzabili da tutti i sottosistemi.

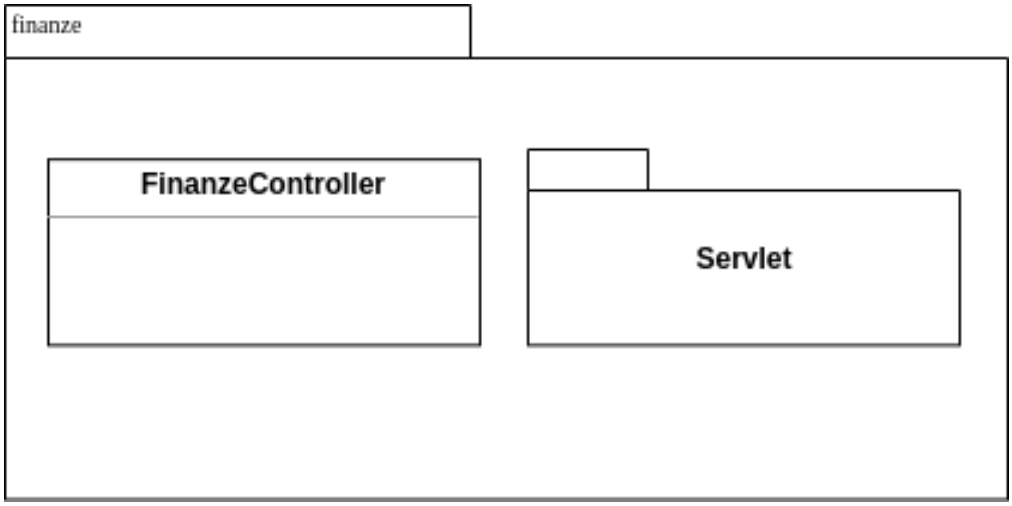
UML Package Diagram



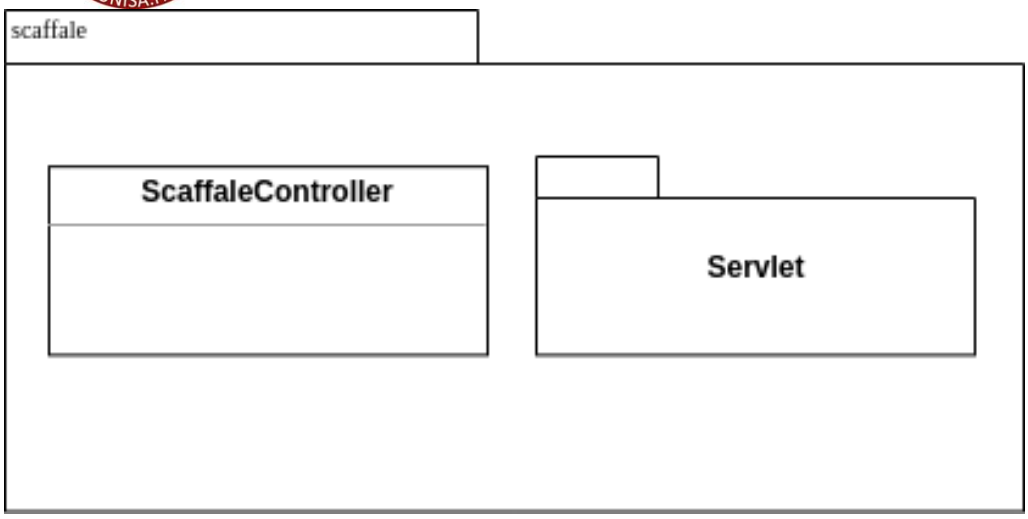
Magazzino



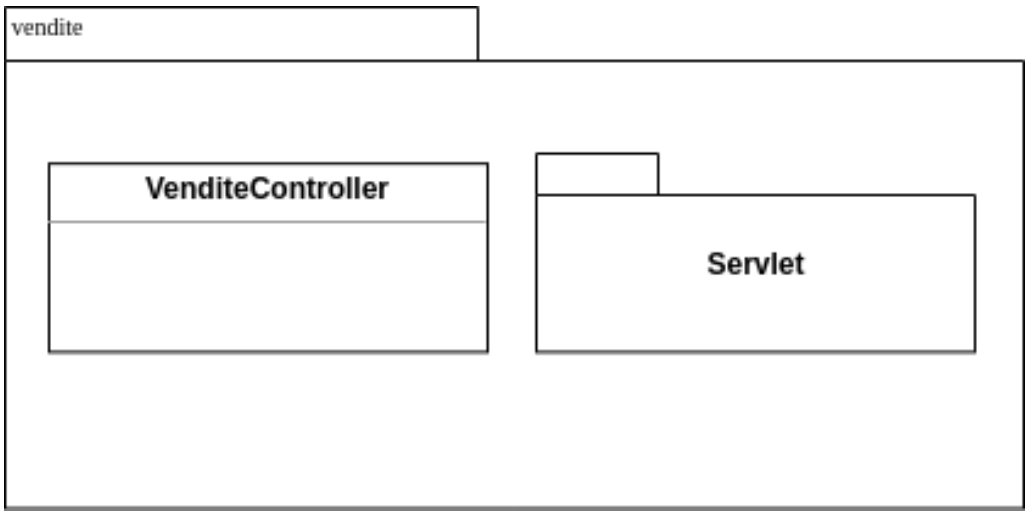
Finanze



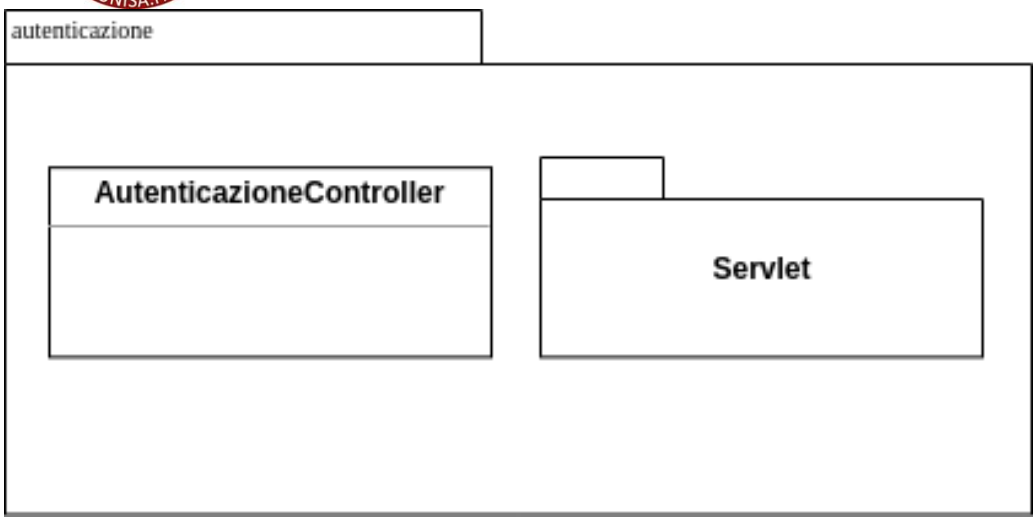
Scaffale



Vendite



Autenticazione



3 Class Interfaces

Di seguito sono riportate le interfacce per ciascun package, in caso di utilizzo di lombok saranno scritte le annotazioni usate per creare i metodi.
La loro documentazione approfondita è specificata tramite **JavaDoc** reperibili nella cartella docs.

3.1 Package Entity

Nome classe	Lotto
Descrizione	La classe in questione descrive il Lotto.
Attributi	-id : int -costo : float -dataScadenza : LocalDate -quantita : int -quantitaAttuale : int -fornitura : Fornitura -prodotto : Prodotto
Metodi	@Getter @Setter @AllArgsConstructor @NoArgsConstructor +getCostoProdotto() : float +getInfo(): String
Invariante di classe	/

Nome classe	Fornitura
Descrizione	La classe in questione descrive la Fornitura.
Attributi	-id : int -giorno : LocalDate -lotti : List<Lotto>
Metodi	@Getter @Setter @AllArgsConstructor
Invariante di classe	/

Nome classe	Prodotto
Descrizione	La classe in questione descrive il Prodotto.
Attributi	-id : int -nome : String -marchio : String -prezzo : float -prezzoScontato : float -inizioSconto : LocalDate -fineSconto : LocalDate -foto : byte[]
Metodi	@Getter @Setter @AllArgsConstructor @NoArgsConstructor +isSconto() : boolean +getInfo() : String
Invariante di classe	/

Nome classe	Esposizione
Descrizione	La classe in questione descrive i prodotti messi in esposizione sugli scaffali.
Attributi	-prodotto : Prodotto -lotto : Lotto -quantita : int
Metodi	@Getter @Setter
Invariante di classe	/

Nome classe	Dipendente
Descrizione	La classe in questione descrive il dipendente
Attributi	-id : int -ruolo : Ruolo -pin : char[]
Metodi	@Getter @Setter @AllArgsConstructor @NoArgsConstructor
Invariante di classe	/

Nome classe	Venduto
Descrizione	Questa classe contiene i dati per rappresentare la vendita.
Attributi	-prodotto : Prodotto -quantita : int -guadagno : float -costo : float -giorno: LocalDate
Metodi	@Getter @Setter
Invariante di classe	/

Nome classe	Bilancio
Descrizione	Questa classe contiene i dati per rappresentare il bilancio.
Attributi	-guadagno : float -spese : float -incasso : float -perdite: float
Metodi	@Getter @Setter +calculateUtile() : float
Invariante di classe	guadagno = incasso - costoProdotto

3.2 Package Magazzino

Nome classe	MagazzinoController
Descrizione	Questa classe effettua operazioni relative al magazzino.
Metodi	+registraFornitura(Fornitura fornitura) : boolean confirm +eliminaLotto(Lotto lotto) : boolean eliminato +modificaFornitura(Fornitura fornitura) : boolean modificato +visualizzaProdottiMagazzino() : HashMap<Prodotto, ArrayList<Lotto>> +visualizzaForniture() : ArrayList<Fornitura> forniture +getSpese(LocalDate dataInizio, LocalDate dataFine) : Float spese +getSpeseTotali() : Float spese +getProdottoById(int id) : Prodotto p +checkProductExists(String nome, String marchio) : Prodotto p +getAllProducts() : List<Prodotto> prodotti
Invariante di classe	/

Nome Metodo	+getSpese(LocalDate dataInizio, LocalDate dataFine) : Float spese
Descrizione	Questo metodo restituisce le spese del supermercato in un periodo scelto
Pre-condizione	**Pre:** dataInizio.isBefore(dataFine) || dataInizio.isEqual(dataFine)
Post-condizione	/

Nome Metodo	+registraFornitura(Fornitura fornitura): boolean confirm
Descrizione	Questo metodo consente di inserire una nuova fornitura
Pre-condizione	**Context**: MagazzinoController::registraFornitura(Fornitura fornitura) **Pre**: !visualizzaForniture().includes(fornitura)
Post-condizione	**Context**: MagazzinoController::registraFornitura(Fornitura fornitura) **Post**: visualizzaForniture().includes(fornitura)
Nome Metodo	+eliminaLotto(Lotto lotto) : boolean eliminato
Descrizione	Questo metodo consente di eliminare un lotto dal supermercato
Pre-condizione	**Context**: MagazzinoController::eliminaLotto(Lotto lotto) **Pre**: visualizzaProdottiMagazzino().includes(lotto)
Post-condizione	**Context**: MagazzinoController::eliminaLotto(Lotto lotto) **Post**: !visualizzaProdottiMagazzino().includes(lotto)
Nome Metodo	+modificaFornitura(Fornitura fornitura, ArrayList<Lotto> lotti) : boolean modificato
Descrizione	Questo metodo modifica la fornitura.

Pre-condizione	/
Post-condizione	**/**
Nome Metodo	+visualizzaProdottiMagazzino() : HashMap<Prodotto, ArrayList<Lotto>> magazzino
Descrizione	Questo metodo consente di visualizzare la lista dei lotti presenti in magazzino
Pre-condizione	**/**
Post-condizione	**/**
Nome Metodo	+visualizzaForniture() : List<Fornitura> forniture
Descrizione	Questo metodo consente di visualizzare tutte le forniture arrivate al suermercato
Pre-condizione	/
Post-condizione	**/**
Nome Metodo	+getProdottoById(int id) : Prodotto p
Descrizione	Questo metodo consente di cercare un determinato prodotto.
Pre-condizione	/
Post-condizione	**/**
Nome Metodo	+checkProductExists(String nome, String marchio) : Prodotto p
Descrizione	Questo metodo consente di verificare se un prodotto, in fase di inserimento, esiste già nella banca dati.
Pre-condizione	/
Post-condizione	**/**

Nome Metodo	+getAllProducts() : List<Prodotto> prodotti
Descrizione	Questo metodo consente di avere tutti i prodotti presenti nella banca dati.
Pre-condizione	/
Post-condizione	**/**

3.3 Package Finanze

Nome classe	FinanzeController
Descrizione	Questa classe effettua operazioni relative alle finanze del supermercato.
Metodi	+visualizzaBilancio() : Bilancio bilancio +visualizzaBilancioParziale(LocalDate inizio, LocalDate fine) : Bilancio bilancio +visualizzaAndamentoProdotto(LocalDate dataInizio, LocalDate dataFine, Prodotto prodotto) : ArrayList<Integer> andamento +impostaSconto(Prodotto prodotto, int sconto, LocalDate dataInizio, LocalDate dataFine): boolean sconto
Invariante di classe	/

Nome Metodo	+visualizzaBilancio() : Bilancio bilancio
Descrizione	Questo metodo consente di visualizzare il bilancio totale del supermercato
Pre-condizione	/
Post-condizione	**/**
Nome Metodo	+visualizzaAndamentoProdotto(LocalDate dataInizio, LocalDate dataFine, Prodotto prodotto) : ArrayList<Integer> andamento
Descrizione	Questo metodo consente di visualizzare l'andamento di un prodotto
Pre-condizione	**Context**: FinanzeController::visualizzaAndamentoProdotto(LocalDate dataInizio, LocalDate dataFine, Prodotto p) **Pre:** dataInizio.isBefore(dataFine) || dataInizio.isEqual(dataFine) && dataFine.isBefore(today)
Post-condizione	**/**
Nome Metodo	+impostaSconto(Prodotto prodotto, int sconto, LocalDate dataInizio, LocalDate dataFine): boolean sconto
Descrizione	Questo metodo consente di impostare uno sconto ad un prodotto
Pre-condizione	**Context**: FinanzeController::impostaSconto(Prodotto prodotto, int sconto, LocalDate dataInizio, LocalDate dataFine) **Pre**: Prodotto.isSconto() == false && dataInizio.isAfter(today) && dataFine.isAfter(dataInizio)

Post-condizione	**Context**: FinanzeController::impostaSconto(Prodotto prodotto, int sconto, LocalDate dataInizio, LocalDate dataFine) **Post**: impostaSconto() == true
Nome Metodo	+visualizzaBilancioParziale(LocalDate inizio, LocalDate fine): Bilancio
Descrizione	Questo metodo consente di visualizzare il bilancio del supermercato per un periodo scelto.
Pre-condizione	**Context**: FinanzeController::visualizzaBilancioParziale(LocalDate dataInizio, LocalDate dataFine) **Pre:** inizio.isBefore(dataFine) || inizio.isEqual(dataFine) && fine.isBefore(today)
Post-condizione	**/**

3.4 Package Scaffale

Nome classe	ScaffaleController
Descrizione	Questa classe effettua operazioni relative agli scaffali.
Metodi	+inserisciEsposizione(int quantita, Lotto l) : void +eliminaScadutiScaffale(): void +visualizzaProdottiScaffale(): ArrayList<Esposizione> prodottiScaffale +visualizzaProdottiScaffaleScaduti(): ArrayList<Esposizione> prodottiEspostiScaduti +visualizzaProdottiMagazzino() : ArrayList<Lotto> lotti +aumentaEsposizione(int qnt, Esposizione e) : void +diminuisciLotto(int id, int qnt): void +getEspostiByProdotto(Prodotto p): int numeroEsposti
Invariante di classe	/

Nome Metodo	+inserisciEsposizione(int quantita, Lotto l): void
Descrizione	Questo metodo consente di aggiungere prodotti di lotti non esposti sullo scaffale.
Pre-condizione	**Context**: inserisciEsposizione(int quantita, Lotto l): void **Post**: !visualizzaProdottiScaffale().includes(l)
Post-condizione	**Context**: inserisciEsposizione(int quantita, Lotto l): void **Post**: visualizzaProdottiScaffale().includes(l)
Nome Metodo	+eliminaScadutiScaffale(): void
Descrizione	Questo metodo consente di eliminare i prodotti scaduti dallo scaffale
Pre-condizione	**Context**: ScaffaleController::eliminaScadutiScaffale() : void **Pre**: visualizzaProdottiScaffale().includes(esposizioneScaduti)
Post-condizione	**Context**: ScaffaleController::eliminaScadutiScaffale() : void **Post**: !visualizzaProdottiScaffale().includes(esposizioneScaduti)
Nome Metodo	+visualizzaProdottiScaffale(): ArrayList<Esposizione> lottiScaffale

Descrizione	Questo metodo consente di visualizzare i prodotti esposti sullo scaffale con le relative quantità
Pre-condizione	**/**
Post-condizione	**/**

Nome Metodo	+visualizzaProdottiScaffaleScaduti(): ArrayList<Esposizione> lottiScaffale
Descrizione	Questo metodo consente di visualizzare i prodotti esposti sullo scaffale ma che sono scaduti
Pre-condizione	**/**
Post-condizione	**/**

Nome Metodo	+visualizzaProdottiMagazzino(): ArrayList<Lotto> lottiScaffale
Descrizione	Questo metodo consente di visualizzare i lotti ancora non esposti sugli scaffali.
Pre-condizione	**/**
Post-condizione	**/**

Nome Metodo	+diminuisciLotto(int id, int qnt): void
Descrizione	Questo metodo consente di diminuire la quantita in magazzino del lotto scelto.
Pre-condizione	**Context**: ScaffaleController::diminuisciLotto() : void **Pre**: Lotto.getQuantitaAttuale() >= qnt && qnt >0
Post-condizione	**Context**: ScaffaleController::diminuisciLotto() : void **Pre**: Lotto.getQuantitaAttuale() = Lotto.getQuantitaAttuale().before - qnt

Nome Metodo	+aumentaEsposizione(int qnt, Esposizione e): void
Descrizione	Questo metodo consente di aumentare la quantita in esposizione di un prodotto.
Pre-condizione	**Context**: ScaffaleController::aumentaEsposizione(int qnt, Esposizione e) : void **Pre**: qnt >0
Post-condizione	**Context**: ScaffaleController::aumentaEsposizione(int qnt, Esposizione e) : void **Post**: e.getQuantita() = e.getQuantita().before() + qnt

Nome Metodo	+getEspostiByProdotto(Prodotto p): int quantita
Descrizione	Questo metodo consente di avere la quantità totale di esposizione di un determinato prodotto a prescindere dal lotto.
Pre-condizione	**/**
Post-condizione	**/**

3.5 Package Vendite

Nome classe	VenditaController
Descrizione	Questa classe effettua operazioni relative alla vendita dei prodotti
Metodi	+venditaProdotti(ArrayList<Venduto> venduti) : boolean conferma +visualizzaStoricoVendite(LocalDate dataInizio, LocalDate dataFine) : ArrayList<Venduto> venduti +visualizzaProdottiEsposti() : ArrayList<Esposizione> esposti +getIncassi(LocalDate dataInizio, LocalDate dataFine) : Float incassi +getIncassiTotali() : Float incassi +getGuadagni (LocalDate dataInizio, LocalDate dataFine) : Float guada +getGuadagniTotali() : Float guadagno +addGiornoVendite() : void
Invariante di classe	/

Nome Metodo	+getIncassi(LocalDate inizio, LocalDate fine) : Float incassi
Descrizione	Questo metodo restituisce gli incassi totali del supermercato dal primo giorno di inaugurazione.
Pre-condizione	**Context:** VenditaController:: getIncassiTotali(LocalDate inizio, LocalDate fine) Pre: inizio.isBefore(fine) && (fine.isBefore(today) || fine.isEqual(today))
Post-condizione	/

Nome Metodo	+getIncassiTotali() : Float incassi
Descrizione	Questo metodo restituisce gli incassi totali del supermercato dal primo giorno di inaugurazione.
Pre-condizione	/
Post-condizione	/

Nome Metodo	+getGuadagni(LocalDate inizio, LocalDate fine) : Float incassi
Descrizione	Questo metodo restituisce i guadagni totali del supermercato in un periodo scelto.
Pre-condizione	**Context:** VenditaController:: getGuadagni(LocalDate inizio, LocalDate fine) Pre: inizio.isBefore(fine) && (fine.isBefore(today) || fine.isEqual(today))
Post-condizione	/

Nome Metodo	+getGuadagniTotali() : Float incassi
Descrizione	Questo metodo restituisce i guadagni totali del supermercato dal primo giorno di inaugurazione.
Pre-condizione	/
Post-condizione	/

Nome Metodo	+visualizzaProdottiEsposti() : ArrayList<Esposizione> esposti
Descrizione	Questo metodo restituisce i prodotti in esposizione raggruppati per prodotto senza distinzione di scadenze.
Pre-condizione	**/**
Post-condizione	/

Nome Metodo	+addGiornoVendite() : void
Descrizione	Questo metodo controlla se il giorno di vendita è gia stato salvato.
Pre-condizione	**/**
Post-condizione	/

Nome Metodo	+venditaProdotti(ArrayList<Venduto> venduti)
Descrizione	Questo metodo consente di confermare le vendite nel supermercato
Pre-condizione	**Context:** VenditaController:: venditaProdotti(Venduto venduto) **Pre**: visualizzaProdottiScaffale().includes(venduto.getProdotto()) && esposizione.getLotto().getQuantitaAttuale() >= venduto.getQuantita
Post-condizione	**Context:** VenditaController:: venditaProdotti(Venduto venduto) **Post:** venditaProdotti(Venduto venduto) == true
Nome Metodo	+visualizzaStoricoVendite(LocalDate dataInizio, LocalDate dataFine) : List<Venduto> venduti
Descrizione	Questo metodo consente di visualizzare le vendite avvenute nel supermercato
Pre-condizione	**Context:** VenditaController::visualizzaStoricoVendite(LocalDate dataInizio, LocalDate dataFine) **Pre:** dataInizio.isBefore(dataFine) || dataInizio.isEqual(dataFine)

Post-condizione	**/**

3.6 Package Autenticazione

Nome classe	AutenticazioneController
Descrizione	Questa classe effettua operazioni relative all'autenticazione dei dipendenti
Metodi	+login(int pin) : Dipendente dipendente +modificaPin(int newPin, Ruolo ruolo) : boolean
Invariante di classe	/

Nome Metodo	+login(int pin) : Dipendente dipendente
Descrizione	Questo metodo verifica il login da parte di un utente
Pre-condizione	/
Post-condizione	/

Nome Metodo	+modificaPin(int newPin, Ruolo ruolo) : boolean pin
Descrizione	Questo metodo consente di modificare il pin di un account
Pre-condizione	**Pre:** Dipendente.getPin ->forAll() != newPin
Post-condizione	**Post:** dipendente.getPin() == newPin

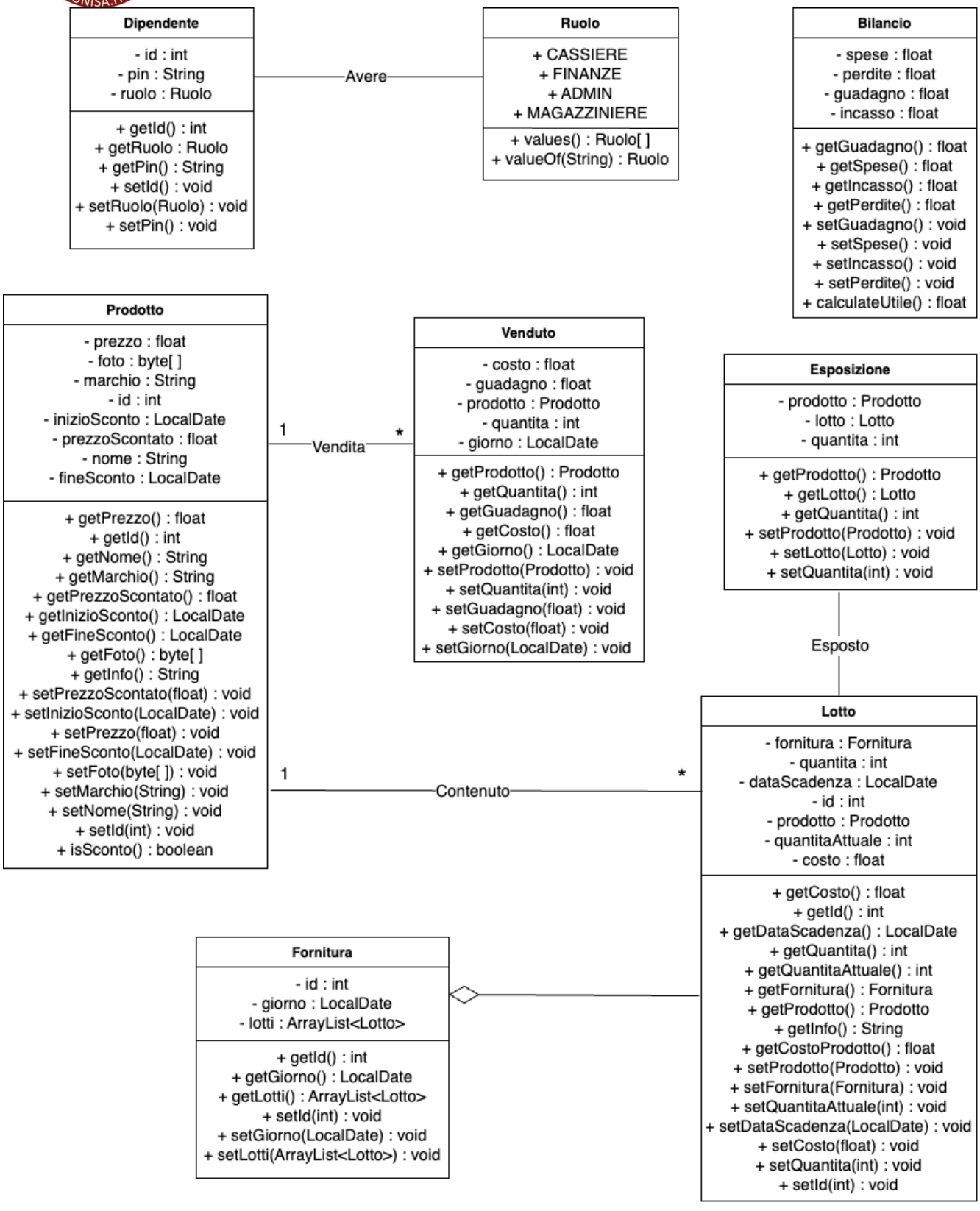
4 Class Diagram Ristrutturato

Prima di passare all'implementazione, il class diagram ottenuto nella fase di analisi è stato ristrutturato per allinearsi con le decisioni prese durante la fase di progettazione.

La gerarchia precedentemente definita, che coinvolgeva la superclasse "Dipendente" e le sue sottoclassi "Cassiere", "Gestore Finanze", "Magazziniere" e "Admin", è stata rimossa. Al suo posto, è stata aggiunta la classe "Dipendente", la quale è stata collegata all'enumerazione "Ruolo" per definire i diversi ruoli che i dipendenti possono assumere nel sistema.

Inoltre, è stata aggiunta la classe "Bilancio" per tenere traccia del bilancio del supermercato.

Infine, le classi "Scaffale" e "Magazzino" sono state rimosse e sostituite da un'unica classe "Esposizione". I lotti in esposizione sono quelli presenti sullo scaffale.


Dipendente
- id : int
- pin : String
- ruolo : Ruolo
+ getId() : int
+ getRuolo : Ruolo
+ getPin() : String
+ setId() : void
+ setRuolo(Ruolo) : void
+ setPin() : void
Avere
Ruolo
+ CASSIERE
+ FINANZE
+ ADMIN
+ MAGAZZINIERE
+ values() : Ruolo[ ]
+ valueOf(String) : Ruolo
Bilancio
- spese : float
- perdite : float
- guadagno : float
- incasso : float
+ getGuadagno() : float
+ getSpese() : float
+ getIncasso() : float
+ getPerdite() : float
+ setGuadagno() : void
+ setSpese() : void
+ setIncasso() : void
+ setPerdite() : void
+ calculateUtile() : float
Prodotto
- prezzo : float
- foto : byte[ ]
- marchio : String
- id : int
- inizioSconto : LocalDate
- prezzoScontato : float
- nome : String
- fineSconto : LocalDate
+ getPrezzo() : float
+ getId() : int
+ getNome() : String
+ getMarchio() : String
+ getPrezzoScontato() : float
+ getInizioSconto() : LocalDate
+ getFineSconto() : LocalDate
+ getFoto() : byte[ ]
+ getInfo() : String
+ setPrezzoScontato(float) : void
+ setInizioSconto(LocalDate) : void
+ setPrezzo(float) : void
+ setFineSconto(LocalDate) : void
+ setFoto(byte[ ]) : void
+ setMarchio(String) : void
+ setNome(String) : void
+ setId(int) : void
+ isSconto() : boolean
1
Vendita
*
Venduto
- costo : float
- guadagno : float
- prodotto : Prodotto
- quantita : int
- giorno : LocalDate
+ getProdotto() : Prodotto
+ getQuantita() : int
+ getGuadagno() : float
+ getCosto() : float
+ getGiorno() : LocalDate
+ setProdotto(Prodotto) : void
+ setQuantita(int) : void
+ setGuadagno(float) : void
+ setCosto(float) : void
+ setGiorno(LocalDate) : void
Esposizione
- prodotto : Prodotto
- lotto : Lotto
- quantita : int
+ getProdotto() : Prodotto
+ getLotto() : Lotto
+ getQuantita() : int
+ setProdotto(Prodotto) : void
+ setLotto(Lotto) : void
+ setQuantita(int) : void
Esposto
1
Contenuto
*
Lotto
- fornitura : Fornitura
- quantita : int
- dataScadenza : LocalDate
- id : int
- prodotto : Prodotto
- quantitaAttuale : int
- costo : float
+ getCosto() : float
+ getId() : int
+ getDataScadenza() : LocalDate
+ getQuantita() : int
+ getQuantitaAttuale() : int
+ getFornitura() : Fornitura
+ getProdotto() : Prodotto
+ getInfo() : String
+ getCostoProdotto() : float
+ setProdotto(Prodotto) : void
+ setFornitura(Fornitura) : void
+ setQuantitaAttuale(int) : void
+ setDataScadenza(LocalDate) : void
+ setCosto(float) : void
+ setQuantita(int) : void
+ setId(int) : void
Fornitura
- id : int
- giorno : LocalDate
- lotti : ArrayList<Lotto>
+ getId() : int
+ getGiorno() : LocalDate
+ getLotti() : ArrayList<Lotto>
+ setId(int) : void
+ setGiorno(LocalDate) : void
+ setLotti(ArrayList<Lotto>) : void

5 Design Patterns

La seguente sezione descrive i design pattern utilizzati nel software.

Per ogni pattern utilizzato, forniremo:

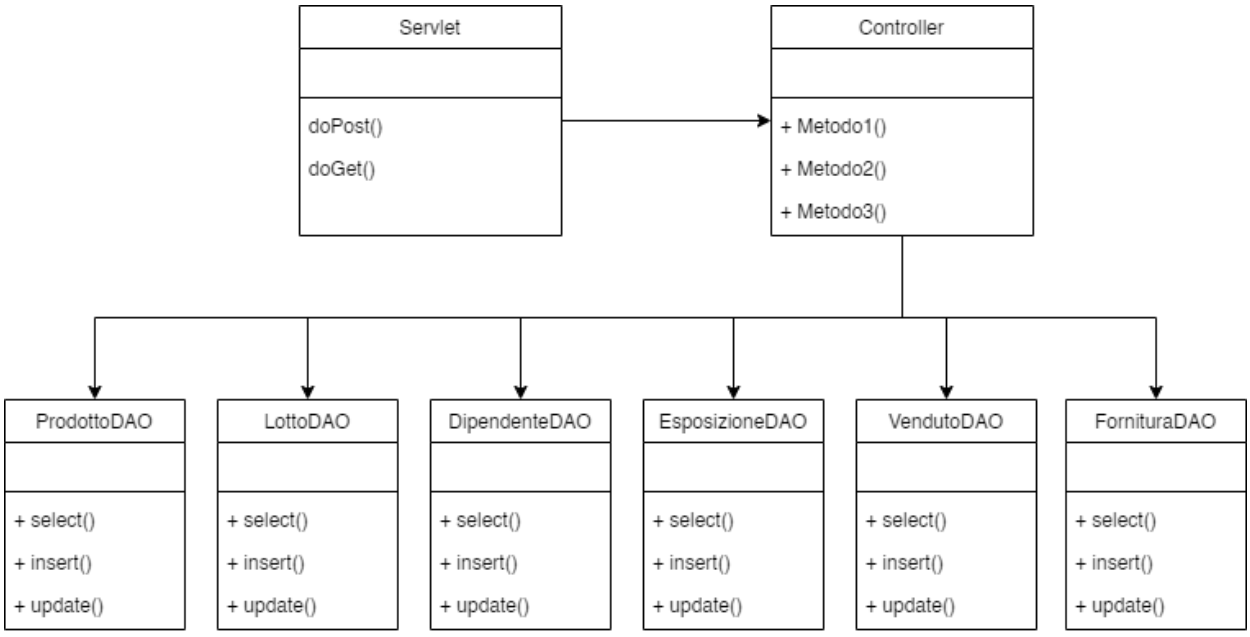
- Spiegazione a livello teorico del pattern.
- Il problema che risolve all'interno del sistema.
- Un diagramma che espone l'implementazione del pattern.

5.1 Facade

Facade è un design pattern che permette di avere un'interfaccia unica semplificata per accedere ad un insieme di operazioni e oggetti che compongono un sottosistema, garantisce un basso accoppiamento e rende la piattaforma più aggiornabile e manutenibile.

Utilizziamo il Facade per ogni sottosistema, quindi esistono 5 classi Facade; nello specifico esse fanno da tramite tra i **DAO**, utili per le interrogazioni al database, e le **servlet**, essenziali per la gestione dei dati ricevuti in input e per l'invio di dati in output alla view. Quindi mettiamo a disposizione delle servlet la logica dei nostri sottosistemi, evitando che esse vadano ad interfacciarsi in modo diretto con le classi DAO.

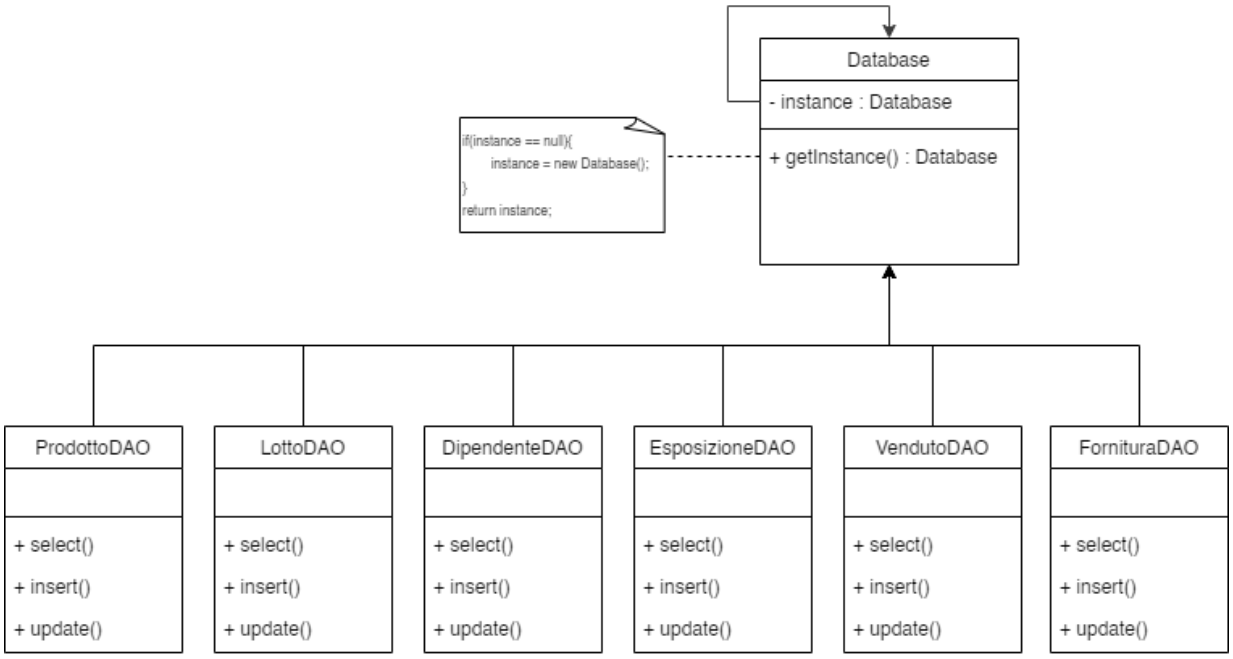
Identifichiamo queste classi con il nome "nome*sottosistemaController*".



5.2 Singleton

Singleton è un design pattern creazionale, ossia un design pattern che si occupa dell'istanziazione degli oggetti, che ha lo scopo di garantire che di una determinata classe venga strutturata una sola istanza e di fornire un punto di accesso globale a tale istanza.

Utilizziamo questo design pattern per gestire la connessione al database, così da garantire che esista un solo ed unico oggetto per ottenere la connessione prima di effettuare una qualsiasi query.



5 Glossario

Sigla/Termine	Definizione
Piattaforma	Base software o hardware sulla quale è sviluppata o in esecuzione l'applicazione.
Lotto	Insieme di stessi prodotti con uguale data di scadenza arrivati da una determinata fornitura.
Fornitura	Rappresentazione in forma digitale dell'arrivo di lotti di vari prodotti con relative quantità scaricate.
Vendita	Riferimento digitale di vendite di prodotti avvenute nel market.
Design pattern	Template di soluzioni a problemi ricorrenti, impiegati per ottenere riuso e flessibilità
Package	Raggruppamento di classi, interfacce o file correlati
Componenti off-the-shelf	Si riferisce a quei componenti o moduli software che sono disponibili già pronti e possono essere acquistati o utilizzati senza richiedere una personalizzazione specifica